## AMENDMENTS TO THE CLAIMS

1. (Currently amended) A computer implemented method of detecting vulnerabilities in a pre-existing source code listing, said source code listing having a listed sequence of expressions, each expression including a set of operands and operators to transform values of the operands, said listed sequence of expressions having an inherent control flow indicative of the run-time execution of the expressions and an inherent data flow indicative of the run-time transformations of operand values, said source code listing further [[and]] having ~~source code variables and~~ routine calls, said routine calls including arguments with which to invoke a routine, said source code listing being stored in a computer-readable medium, said computer implemented method comprising the acts of:

executing computer instructions to analyze ~~at least one of the inherent control flow and inherent data flow of~~ the source code listing ~~to analyze variables in the source code listing in the context of at least one of the inherent control flow and inherent data flow~~ to create computer models ~~therefrom~~ of the operands, said models each including a corresponding initial set of information to represent the range of values for said operand, and said models being transformed, in response to analysis of the source code, to have a transformed range of values to correspond to the operand transformations expressed in the source code listing, said models being stored in computer memory, ~~in which~~ and wherein each model specifies pre-determined characteristics about and possible values for each operand ~~variable~~ as a result of said ~~control flow and data flow~~ source code expressions;

executing computer instructions to use said operand ~~variable~~ models to create models of said arguments to routine calls, said argument models being stored in computer memory;

executing computer instructions to use said argument models in conjunction with pre-specified criteria for the corresponding routine calls to determine whether the routine calls possess vulnerabilities as a consequence of the arguments and known routine behavior; and

generating a report that identifies the vulnerabilities said report being viewable by a developer-user, so the developer-user may address the vulnerabilities identified in the report by modifying the source code listing if necessary.

2. (Previously presented) The computer implemented method of claim 1 wherein the models specify the memory size of a variable.

3. (Previously presented) The computer implemented method of claim 1 wherein the models specify the data size of a variable.

4. (Previously presented) The computer implemented method of claim 1 wherein the models specify whether the variable is a null terminated string or not null terminated string for variables of string value type.

5. (Previously presented) The computer implemented method of claim 1 wherein the models specify the type of memory of the variable.

6. (Previously presented) The computer implemented method of claim 1 wherein the models specify the value of a string for variables that are of string value type.

7. (Previously presented) The computer implemented method of claim 1 wherein the models specify the origin of the data for a variable.

8. (Previously presented) The computer implemented method of claim 1 wherein the argument models specify characteristics of variable arguments.

9. (Previously presented) The computer implemented method of claim 1 wherein the argument models specify characteristics of expression arguments.

10. (Previously presented) The computer implemented method of claim 1 wherein the models are specified as lattices.

11. (Previously presented) The computer implemented method of claim 10 wherein the lattice values include at least one of a value to represent no knowledge, a value to represent inconsistent knowledge, and a value to represent a refinement of knowledge.

12. (Previously presented) The computer implemented method of claim 11 wherein the value to represent a refinement of knowledge includes values to specify a range of specific values.

13. (Currently amended) The computer implemented method of claim 1 wherein the pre-specified criteria for the corresponding routine includes computer-encoded rules about the semantic behavior of the routine.

14. (Previously presented) The computer implemented method of claim 1 wherein the vulnerabilities are buffer overflows.

15. (Currently amended) A computer implemented method of detecting vulnerabilities in a pre-existing source code listing, said source code listing having a listed sequence of expressions, each expression including a set of operands and operators to transform values of the operands, said listed sequence of expressions having an inherent control flow indicative of the run-time execution of the expressions and an inherent data flow indicative of the run-time transformations of operand values, said source code listing further [[and]] having ~~source code variables and~~ routine calls, said routine calls including arguments with which to invoke a routine said source code listing being stored in a computer-readable medium, said computer implemented method comprising the acts of:

executing computer instructions to analyze ~~at least one of the inherent control flow and inherent data flow of~~ the source code listing ~~to analyze variables in the source code listing in the context of at least one of the inherent control flow and inherent data flow~~ to create computer models of arguments to routine calls in the source code listing, said argument models being stored in computer memory, said argument models each including a corresponding initial set of information to represent the range of values for said argument, and said argument models being transformed, in response to analysis of the source code, to have a transformed range of values to correspond to the transformations expressed in the source code listing, said models being stored in computer memory, and wherein each argument model specifies pre-determined characteristics about and possible values for each argument as a result of said source code expressions;

executing computer instructions to use said argument models in conjunction with pre-specified criteria for the corresponding routine calls to determine whether the routine

calls possess vulnerabilities as a consequence of the arguments and the routine behavior; and

generating a report that identifies the vulnerabilities said report being viewable by a developer-user, so the developer-user may address the vulnerabilities identified in the report by modifying the source code listing if necessary.

16. (Currently amended) A computer implemented utility for detecting vulnerabilities in a pre-existing source code listing, said source code listing having a listed sequence of expressions, each expression including a set of operands and operators to transform values of the operands, said listed sequence of expressions having an inherent control flow indicative of the run-time execution of the expressions and an inherent data flow indicative of the run-time transformations of operand values, said source code listing further [[and]] having ~~source code variables and~~ routine calls, said routine calls including arguments with which to invoke a routine, said source code listing being stored in a computer-readable medium, said utility comprising a computer-readable medium encoded with:

executable instructions for analyzing ~~at least one of the inherent control flow and inherent data flow of~~ the source code listing ~~to analyze variables in the source code listing in the context of at least one of the inherent control flow and data flow and for creating~~ to create computer models of the operands, said models each including a corresponding initial set of information to represent the range of values for said operand, and said models being transformable, in response to analysis of the source code, to have a transformed range of values to correspond

to the operand transformations expressed in the source code listing, said models ~~therefrom~~, storable in a computer memory, ~~in which~~ and wherein each model specifies pre-determined characteristics about and possible values for each operand ~~variable~~ as a result of said ~~control flow and data flow~~ source code expressions;

executable instructions for using the ~~variable~~operand models to create models of arguments to routine calls in the source code listing, said argument models being stored in computer memory; and

executable instructions for using the argument models in conjunction with pre-specified criteria for the corresponding routine calls to determine whether the routine calls possess vulnerabilities as a consequence of the arguments and known routine behavior; and

executable instructions for generating a report that identifies the vulnerabilities said report being viewable by a developer-user, so the developer-user may address the vulnerabilities identified in the report by modifying the source code listing if necessary.

17. (Previously presented) The computer implemented utility of claim 16, using a data base having computer readable information about a predefined set of source code routine calls, said information specifying one or more conditions that present a vulnerability during execution of the source code routine call, wherein the executable instructions for using the argument models in conjunction with pre-specified criteria for the corresponding routine calls to determine whether the routine calls possess vulnerabilities as a consequence of the arguments and known routine behavior includes executable instructions for using the

database to retrieve information for a corresponding routine call to check for the specified condition to see whether the routine call presents a vulnerability.

18. (Previously presented) The computer implemented method of claim 1, using a database having computer-readable information about a predefined set of source code routine calls, said information specifying one or more conditions that present a vulnerability during execution of the source code routine call, wherein the act of using the argument models in conjunction with pre-specified criteria for the corresponding routine calls to determine whether the routine calls possess vulnerabilities as a consequence of the arguments and known routine behavior comprises the act of using the data base to retrieve information for a corresponding routine call to check for the condition to see whether the routine call presents vulnerability.

19. (Previously presented) The computer implemented method of claim 15, using a database having computer-readable information about a predefined set of source code routine calls, said information specifying one or more conditions that present a vulnerability during execution of the source code routine call, wherein the act of using the argument models in conjunction with pre-specified criteria for the corresponding routine calls to determine whether the routine calls possess vulnerabilities as a consequence of the arguments and the routine behavior comprises the act of using the data base to retrieve information for a corresponding routine call to check for the condition to see whether the routine call presents a vulnerability.

20. (Previously presented) The computer implemented method of claim 1 wherein the report identifies the location in the source code listing where the vulnerability occurred.

21. (Previously presented) The computer implemented method of claim 15 wherein the report identifies the location in the source code listing where the vulnerability occurred.

22. (Previously presented) The computer implemented utility of claim 16 wherein the report identifies the location in the source code listing where the vulnerability occurred.